CMPT 280

Topic 4: Timing Analysis

Mark G. Eramian

University of Saskatchewan

References

- Textbook, Chapter 4

Timing Analysis

Express the time required by an algorithm for an input of size n in Big-Oh notation. Also referred to as find the *time complexity* of an algorithm.

1. Determine time required as a function of the input size n (may use best-case or worst-case analysis).
   - Statement counting
   - Active operation
2. Simplify the resulting function to Big-Oh notation.
   - By inspection
   - By formal proof
3. If best and worst case analysis yield same result, may write as Big-Theta.

Exercise 1

Consider this method which returns true if the array a contains the string s.

```
public static bool contains(String[ ] a, String s)
{
    int count = 0;
    int i = 0;
    while (i < a.length && !s.equals(a[i]))
    {
        i = i + 1;
    }
    return i < a.length;
}
```

true n times
false 1 times

a) What is the best case?

b) What is the worst case?

c) Exactly how many statements are executed in the worst case? Best case?

2*n+4

4

Exercise 2

inner loop: 3*i+1

outer loop: 3*i+1+5

worst case: # of iteration:

n-1(outer loop)

sum from 1 to n-1:3i+6 +1(total # or)

3n(n-1)/2+6(n-1)+2(int i=1)

```
public static <T extends Comparable<T>>
            void insertionSort(T[ ] a)
{
    // For each string in the array...
    int i = 1;
    while (i < a.length)
    {
        // Get the i-th element.
        T  temp = a[i];

        // Examine all of the elements that come before it, starting at
        // the rightmost.  If 'temp' comes before a[j], move a[j] one
        // index to the right and continue.

        int j = i - 1;
        while (j >= 0 && temp.compareTo(a[j]) < 0)
        {
            a[j+1] = a[j];
            j = j - 1;
        }

        // We've found the insertion point.  Copy the string here
        // and advance to the next insertion.
        a[j+1] = temp;
        i = i + 1;
    }
}
```

- Exactly how many statements are executed in the worst case?

Exercise 3

- Repeat exercise 1 and 2 using the active operation approach. Exactly how many times does the active operation get executed?

Exercise 4

What is the Big-Oh membership of each of the following functions? We would like the tightest upper bound. Simplify by inspection.

- n
- $47n \log n + 10000n$
- $100n + 500 \log n + 1000$
- $\log n + 100\sqrt{n} + 76$
- n^2
- 5 1
- $2^n + n + \log n$
- $T^B_{contains}(n) = 4$
- $T^W_{contains}(n) = 2n + 4$
- $T^W_{insort}(n) = 1.5n^2 + 4.5n - 4$

2^n > n^2

A More Elaborate Example

Suppose method $q()$ calls another method `int k(int i)` with $T_k(m) = O(\log(m))$ for some m independent of i.

O(n log(m))

```
public void q()
{
    int c = 0;
    for (int i = 1; i < n + 1; i++)
        c = c + k(i);
}
```

Determine the time complexity (i.e. the time execution time as a function of input size expressed in Big-Oh notation) of this method using the active operation approach.

- What is the active operation?
- How many times does it execute?
- What is the cost of the active operation?
- What is the total amount of time required expressed in Big-Oh notation.

Exercise 5

Again, assume method `k` requires $O(\log m)$ time, m independent of i and n.

cost of A.O: 2*O(log m)

```
public void s()
{
    int x = 0;
    for (int i = 0; i < n + 1; i++)
        x = x + k(i)*k(2*n - i);
    System.out.println(x);
}
```

(n+1)* 2*O(log(m))
=nO(log(m))

2nO(log(m))+2*O(log(m))
=nO(log(m))
=O(n log(m))

- What is the active operation?
- How many times does it execute?
- What is the cost of the active operation?
- What is the total amount of time required expressed in Big-Oh notation.

k()+k(): O(log m)
k()*k(): 2*O(log m)
k()^k():

Exercise 6

assume r execute:
n/2^r-1 (p>1) n/2^r (false)
n/2^r <= 1 < n/2^r-1
log(n) <= r, r-1 > log(n)
# of execution: [(log(n) +1)]
cost of A.O: O(1)
r = log(n)

```java
static public int f(int n)
{
    int result = 0;
    double p =n;
    while (p > 1)
    {
        p = p/2;
        result = result + 1;
    }
    return result;
}
```

What is the time complexity of the method?

Combining Growth Functions

- Since $O(f(n))$ and $O(g(n))$ are sets, we can use these set properties and the definition of Big-Oh to show:

$$\begin{aligned}
O(f(n)) + O(g(n)) &= O(f(n) + g(n)) \\
&= O(2 \cdot \max(f(n), g(n))) \\
&= O(\max(f(n), g(n))) \\
O(f(n)) \cdot O(g(n)) &= O(f(n) \cdot g(n)) \\
kO(f(n)) &= O(kf(n)) = O(f(n)) \text{ for any constant } k.
\end{aligned}$$

- Same holds for Big-Theta. We will need at least one of these "rules" to solve our next problem.

Exercise 7

line 3: O(n log(n))
line 6 : O(m log(m))
line 5 :
# of execution:
O(log(n))
cost of A.O: O(n^2)
total: O(n^2 log(m))

```
1 public void r()
2 {
3     q();
4     for (int i = 1; i < m; i = 2*i)
5         p();
6     s();
7 }
```

- Assuming that we know that:
    - $T_q(n,m) \in O(n \log m)$
    - $T_p(n) \in O(n^2)$
    - $T_s(m) \in O(m \log m)$

  What is the time complexity of the method `r`?

r(): O(m log(m) + n^2 log(m)) (unknown m, n^2)
=O(max(m log(m) + n^2 log(m)))

  Use the active operation approach. There are three candidate active operations, what are they? Determine time required by all three; the one that requires the most time is reflective of the overall time complexity.

Exercise 8

- Now we have the tools to return to our list class and consider the time complexity of our list operations.
- Determine the worst-case time complexity of all of the methods in the LinkedList class.
- Are there any methods where the time complexity could be made better?

Exercise 9

Although we now know how to convert an expression into Big-Oh notation by inspection, we should also practice the formal proofs.

Recall definition of Big-Oh notation:

$t(n) \in O(f(n))$ if there exists constants $c, n_0 > 0$ such that $t(n) \leq cf(n)$ for all $n \geq n_0$.

a) Prove that the function $5n^3 + 4\log n + 11n$ is $O(n^3)$.

b) Prove that the function $2n\log n + 4\log n$ is $O(n\log n)$.

Do Growth Rates Matter?

- Does the growth rate make a difference, or are computers so fast that any algorithm can be done quickly?
- Some algorithms take huge amounts of time on even small problems.
- Linear vs. quadratic alone is big difference.

Do Growth Rates Matter?

Time to execute $f(n)$ operations at 10^6 operations per second.

n	$\log_2 n$	n	$n \log_2 n$	n^2	2^n
10	$3.3 \times 10^{-6}s$	$1.0 \times 10^{-5}s$	$3.3 \times 10^{-5}s$	$1.0 \times 10^{-4}s$	$1.0 \times 10^{-3}s$
100	$6.6 \times 10^{-6}s$	$1.0 \times 10^{-4}s$	$6.6 \times 10^{-4}s$	$1.0 \times 10^{-2}s$	$4.0 \times 10^{16}y^*$
1000	$9.97 \times 10^{-6}s$	$1.0 \times 10^{-3}s$	$9.97 \times 10^{-3}s$	$1.0s$	
10^5	$1.66 \times 10^{-5}s$	$0.1s$	$1.66s$	$2.78h$	
10^6	$1.99 \times 10^{-5}s$	$1s$	$19.9s$	$11.57d$	
10^7	$2.33 \times 10^{-5}s$	$10s$	$233s$	$3.17y$	

Do Growth Rates Matter?

Time to execute $f(n)$ operations at 10^6 operations per second.

n	$\log_2 n$	n	$n \log_2 n$	n^2	2^n
10	$3.3 \times 10^{-6}s$	$1.0 \times 10^{-5}s$	$3.3 \times 10^{-5}s$	$1.0 \times 10^{-4}s$	$1.0 \times 10^{-3}s$
100	$6.6 \times 10^{-6}s$	$1.0 \times 10^{-4}s$	$6.6 \times 10^{-4}s$	$1.0 \times 10^{-2}s$	$4.0 \times 10^{16}y^*$
1000	$9.97 \times 10^{-6}s$	$1.0 \times 10^{-3}s$	$9.97 \times 10^{-3}s$	$1.0s$	
10^5	$1.66 \times 10^{-5}s$	$0.1s$	$1.66s$	$2.78h$	Why
10^6	$1.99 \times 10^{-5}s$	$1s$	$19.9s$	$11.57d$	bother?
10^7	$2.33 \times 10^{-5}s$	$10s$	$233s$	$3.17y$	

* Est. age of universe: $1.3 * 10^{10}$ years.

Next Class

- Next class reading: Chapter 5: Abstract Data Types.